Scalability Issues in Sparse Factorization and Triangular Solution

Sherry Li

Lawrence Berkeley National Laboratory

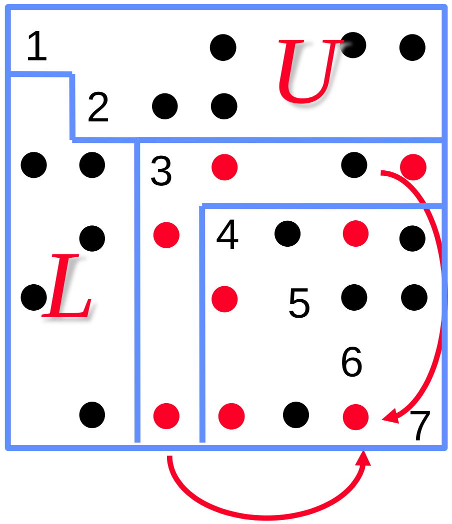
Sparse Days, CERFACS, June 23-24, 2008

Overview

- Basic algorithms
- Partitioning, processor assignment at different phases
- Scalability issues
- Current & future work

Sparse GE

- Scalar algorithm: 3 nested loops
  - Can re-arrange loops to get different variants: left-looking, right-looking, . . .




```
for i = 1 to n
    column_scale ( A(:,i) )
    for k = i+1 to n  s.t.  A(i,k) != 0
        for j = i+1 to n  s.t.  A(j,i) != 0
            A(j,k) = A(j,k) - A(j,i) * A(i,k)
```

- ➢ Typical fill-ratio: 10x for 2D problems, 30-50x for 3D problems
- ➢ Finding fill-ins is equivalent to finding **transitive closure** of G(A)

Major stages

1. Order equations & variables to preserve sparsity
   - NP-hard, use heuristics
2. Symbolic factorization
   - Identify supernodes, set up data structures and allocate memory for L & U.
3. Numerical factorization – usually dominates total time
   - How to pivot?
4. Triangular solutions – usually less than 5% total time

SuperLU_MT

1. Sparsity ordering
2. Factorization
   - Partial pivoting
   - Symbolic fact.
   - Num. fact. (BLAS 2.5)
3. Solve

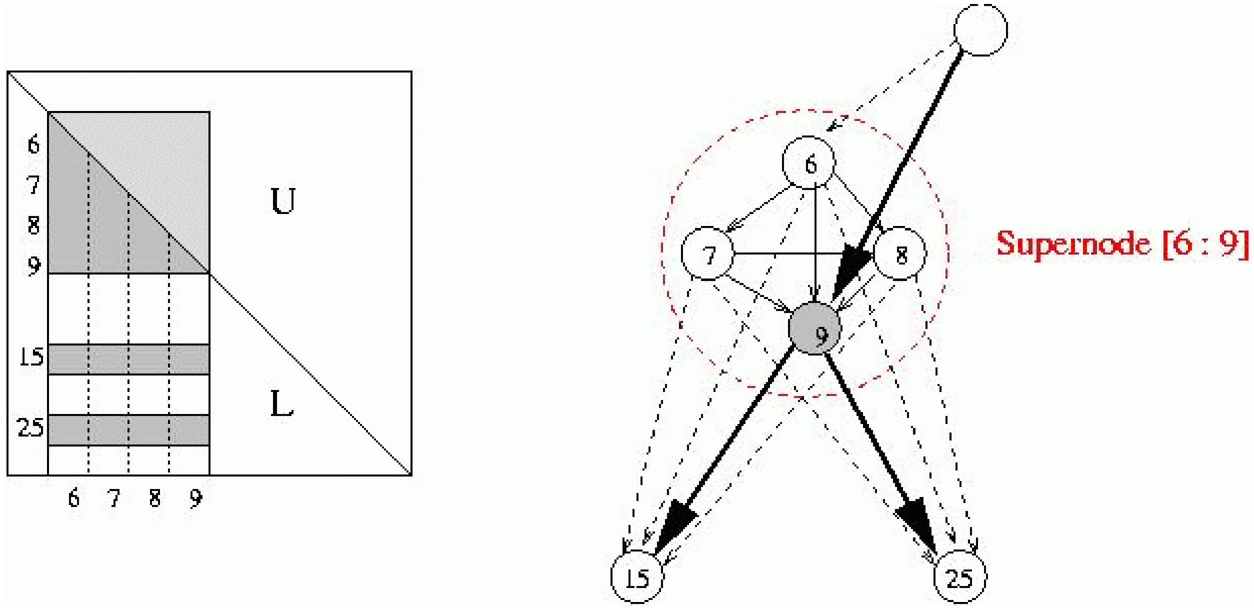
SuperLU_DIST

1. Static pivoting
2. Sparsity ordering
3. Symbolic fact.
4. Numerical fact. (BLAS 3)
5. Solve

SuperLU_DIST steps:

- Static numerical pivoting: improve diagonal dominance
  - **Currently use MC64 (HSL, serial)**
  - **Being parallelized [J. Riedy]: auction algorithm**
- Ordering to preserve sparsity
  - **Can use parallel graph partitioning: ParMetis, Scotch**
- Symbolic factorization: determine pattern of {L\U}
  - **Parallelized [L. Grigori et al.]**
  - Numerics: **Parallelized**
  - **Factorization: usually dominate total time**
  - **Triangular solutions**
  - **Iterative refinement: triangular solution + SPMV**

Supernode: dense blocks in {L\U}



- Good for high performance
  - **Enable use of BLAS 3**
  - **Reduce inefficient indirect addressing (scatter/gather)**
  - **Reduce time of the graph algorithms by traversing a coarser graph**

Matrix partitioning at different stages

- Distributed input A (user interface)
  - **1-D block partition (distributed CRS format)**
- Parallel symbolic factorization
  - **Tied with a ND ordering**
  - **Distribution using separator tree, 1-D within separators**
- Numeric phases
  - **2-D block cyclic distribution**

Parallel symbolic factorization

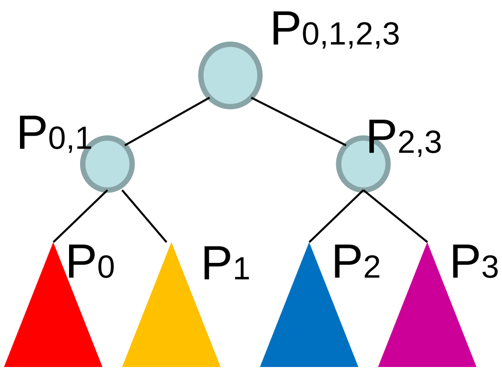
- Tree-based partitioning / assignment

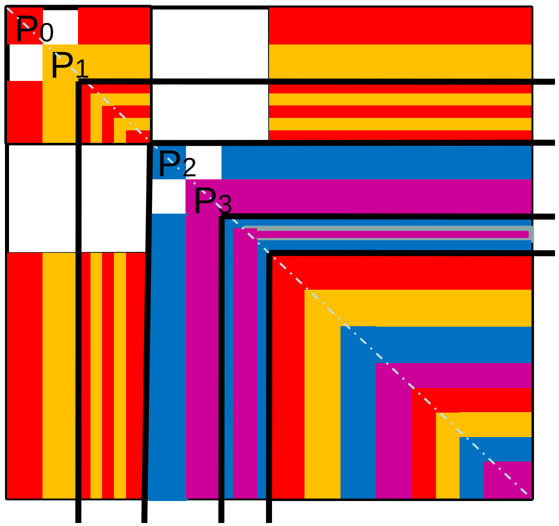
- Use graph partitioning to reorder/partition matrix
  - **ParMetis on graph of A + A'**

- `Arrow-head', two-level partitioning
  - **separator tree: subtree-to-subprocessor mapping**
  - **within separators: 1-D block cyclic distribution**

- **Disadvantage:**

  works only with ND ordering, and a binary tree

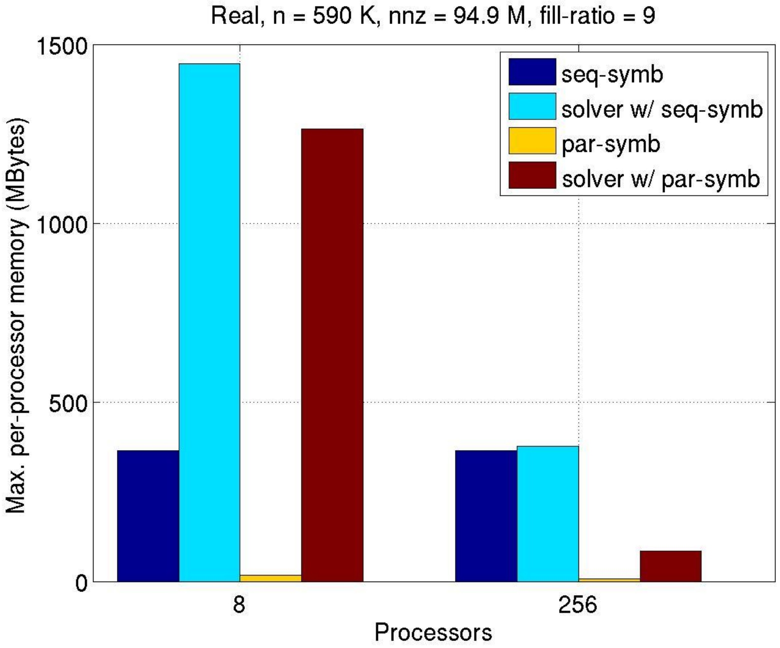




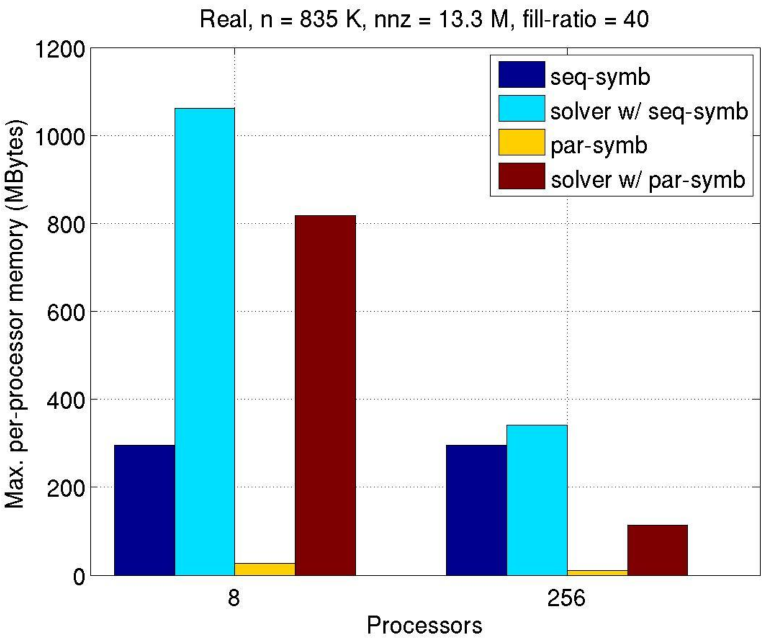
Memory result of parallel symbolic

- Maximum per-processor memory

Fusion: matrix181 (*M3D-C1*)



Accelerator: dds15 (*Omega3P*)



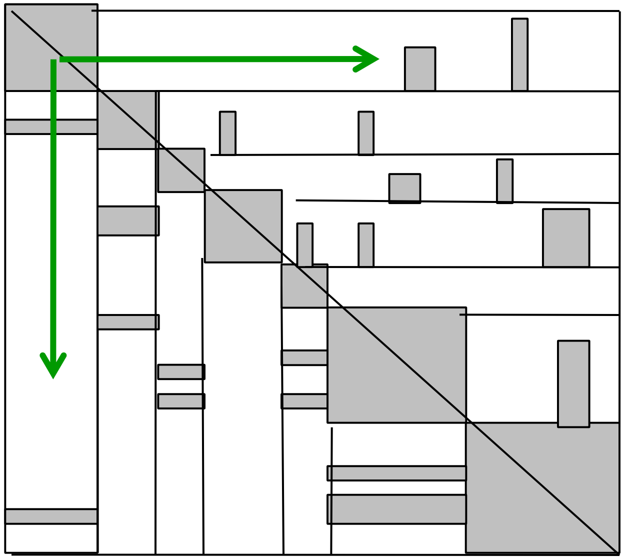
Runtime of parallel symbolic, IBM Power5

matrix181		P = 8	P = 256
symbolic	Sequntial Parallel	6.8 2.6	6.8 2.7
Entire solver	Old New	84.7 159.2	26.6 26.5

dds15		P = 8	P = 256
symbolic	Sequntial Parallel	4.6 1.6	4.6 0.5
Entire solver	Old New	64.1 66.3	43.2 31.4

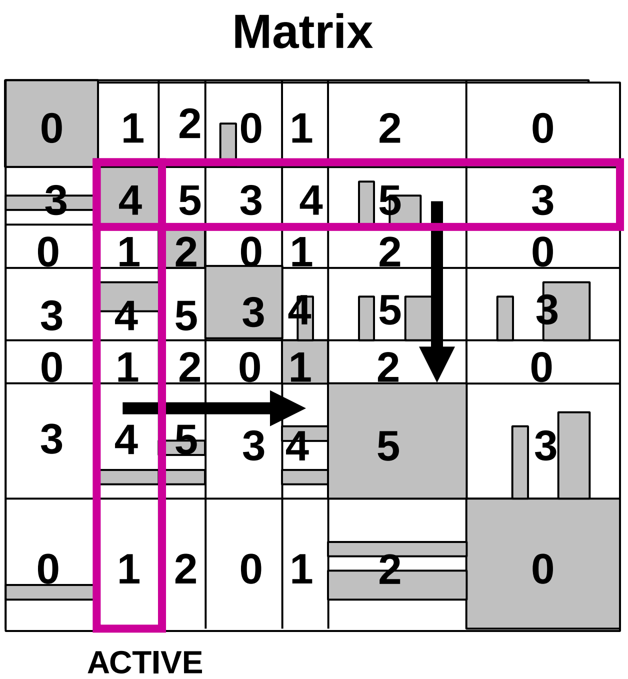
Numeric phases: 2-D partition by supernodes

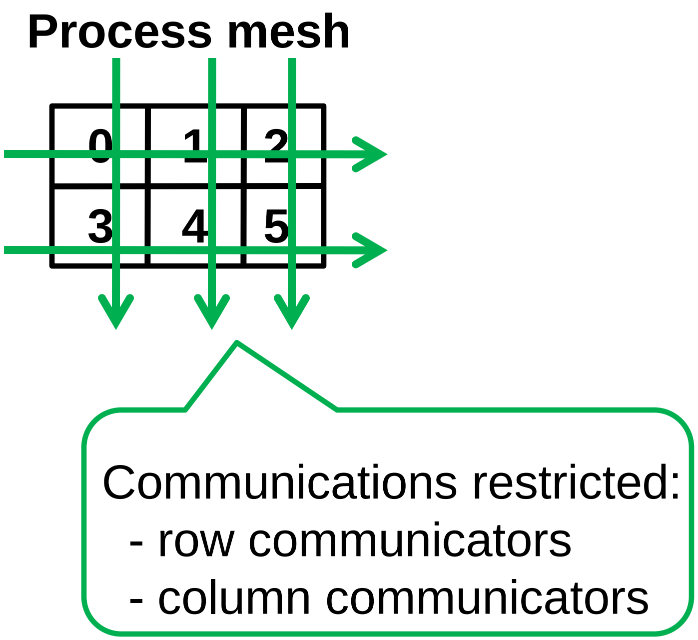
- Find supernode boundaries from columns of L
  - **Not to exceed MAXSUPER (~50)**
- Apply same partition to rows of U
- Diagonal blocks are square, full, <= MAXSUPER; off-diagonal blocks are rectangular, not full



Processor assignment in 2-D

- 2D block cyclic layout
- One step look-ahead to overlap comm. & comp.
- Scales to 1000s processors

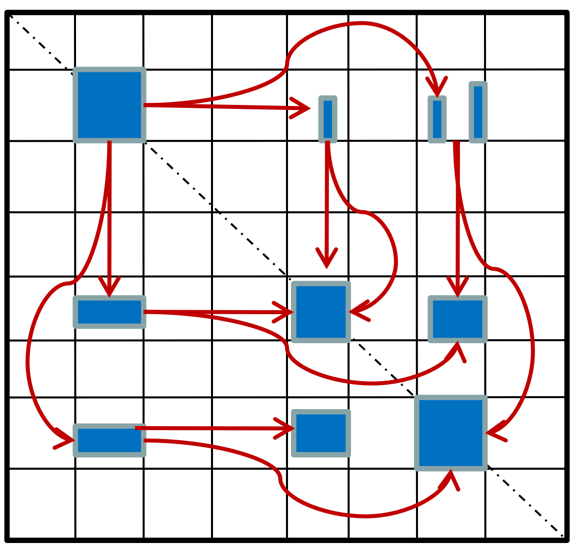




➢Disadvantage: inflexible

Block dependency graph – DAG

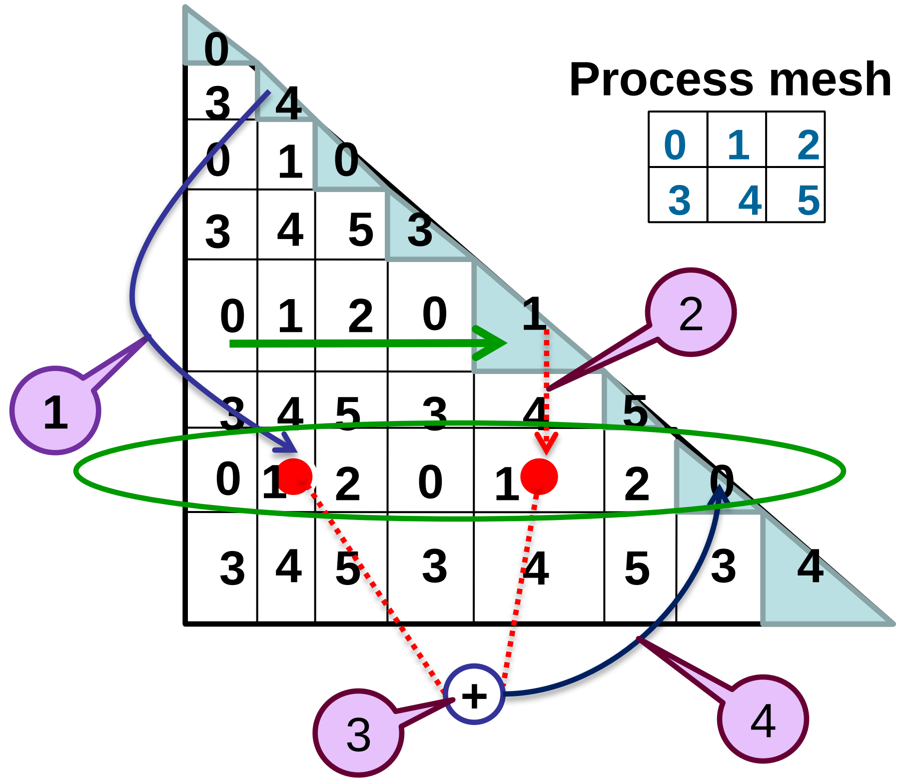
- Based on nonzero structure of L+U
  - **Each diagonal block has edges directed to the blocks below in the same column (L-part), and the blocks on the right in the same row (U-part)**
  - **Each pair of blocks L(r,k) and U(k,c) have edges directed to block (r,c) for Schur complement update**



➢Elimination proceeds from source to sink
➢Over the iteration space
for k = 1 : N,
dags and submatrices become smaller

Triangular solution

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij} \cdot x_j}{L_{ii}}$$



- Higher level of dependency
- Lower arithmetic intensity (flops per byte of DRAM access or communication)

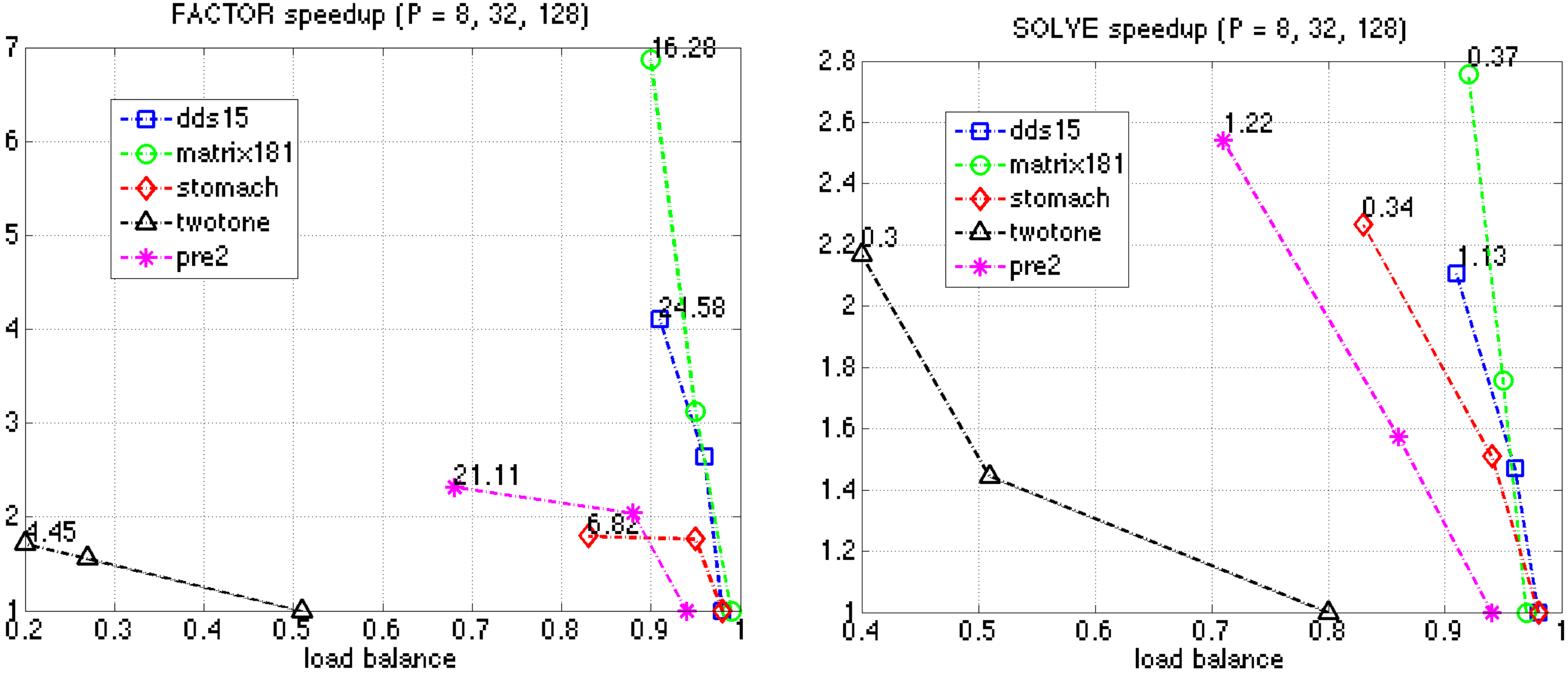
Examples

Name	Codes	N	|A| / N	Fill-ratio
dds15	Acclerator (Omega3P)	834,575	16	40.2
matrix181	Fusion (M3D-C1)	589,698	161	9.3
stomach	3D finite diff.	213,360	14	45.5
twotone	Nonlinear anal. circuit	120,750	10	9.3
pre2	Circuit in Freq.-domain	659,033	9	18.8

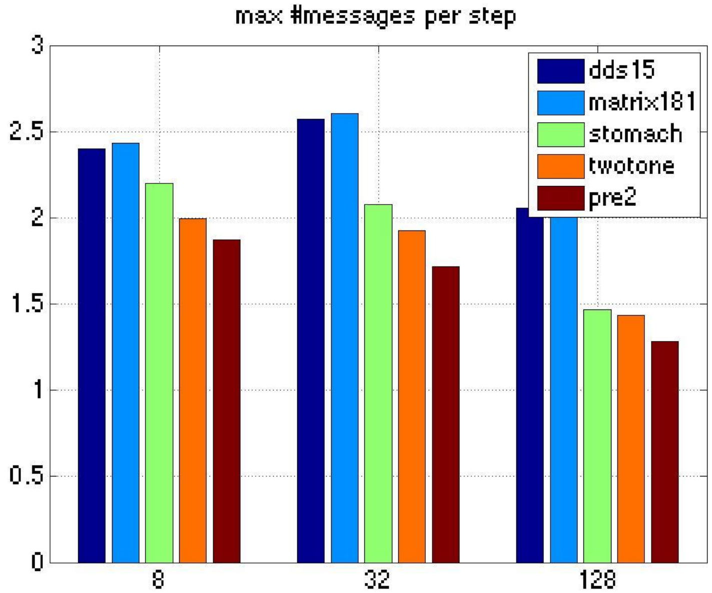
- Sparsity-preserving ordering: MMD applied to structure of A'+A

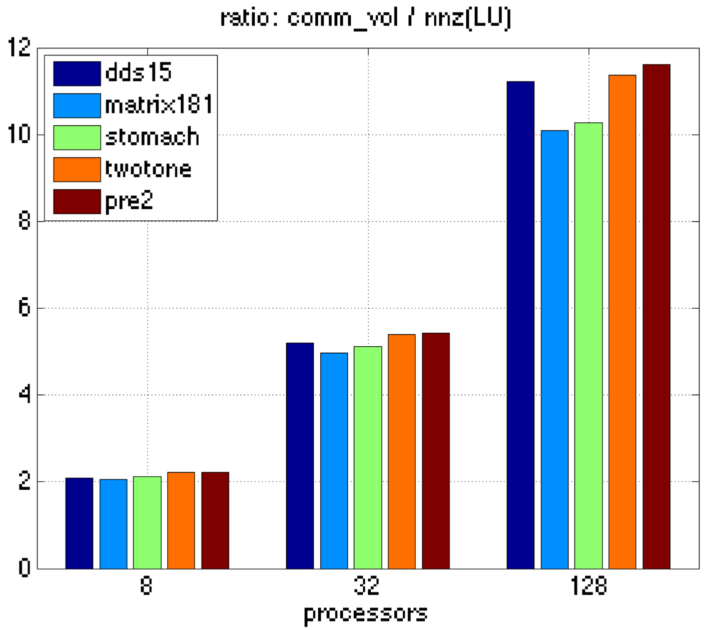
Load imbalance

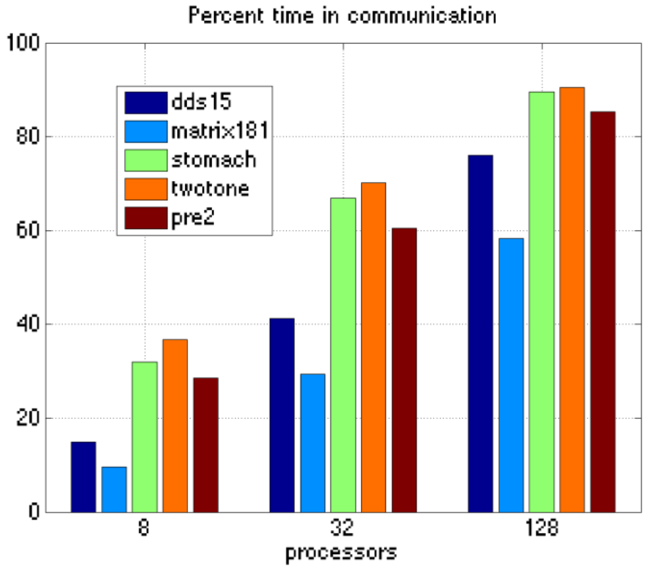
- LB = avg-flops / max-flops



Communication

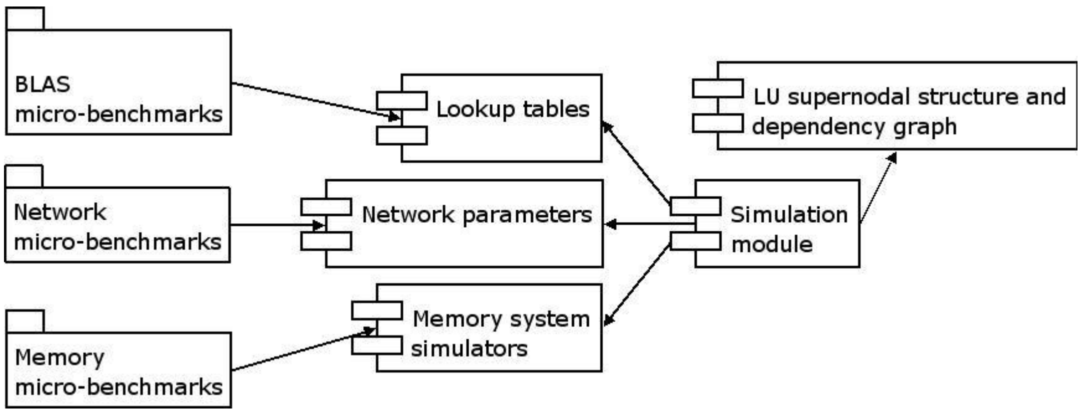






Current and future work

- **LUsim – simulation-based performance model [P. Cicotti et al.]**
  - **Micro-benchmarks to calibrate memory access time, BLAS speed, and network speed**
  - **Memory system simulator for each processor**
  - **Block dependency graph**



- **Better partition to improve load balance**
- **Better scheduling to reduce processor idle time**